Spatter

This is the OLCF-6 memory pattern benchmark.

Author: Nick Hagerty

This repository describes the memory pattern benchmark from the OLCF-6 Benchmark Suite. The OLCF-6 benchmark run rules should be reviewed before running this benchmark.

Note, in particular:

- The OLCF-6 run For this benchmark, ricitly noted within this README
- For this benchmark, responses should only include performance estimates for the "ported" category. Algorithm changes are not permitted to Spatter.
- Responses should estimate the performance for the target problem on the target architecture. The projected walltime for the target problem on the target system must not exceed the reference time measured by running the reference problem on Summit.

Memory Pattern Overview

Spatter is a pure-memory microbenchmark for timing Gather/Scatter memory access patterns on CPUs (Serial, OpenMP) and GPUs (CUDA). This code is hosted on GitHub at https://github.com/hpcgarage/spatter (https://github.com/hpcgarage/spatter). Spatter receives a memory pattern from the command-line argument `-p`, which may provide either a sequence of indexes or the path to a JSON file containing one of more sequences of indexes. In this repository we supply a JSON file that contains 1 or more memory patterns.

The application codes XRAGE (LANL) informed the development of 9 memory patterns captured by the provided JSON file. The Spatter microbenchmark runs Gather or Scater memory access patterns, which target read (Gather) and write (Scatter) memory characteristics. For example, a Gather access kernel could read every 8th entry from a source array into local variables. A Scatter pattern would be the opposite -- writing every entry in a source array into every 8th position in an array that is at least 8x larger.

Pure-memory benchmarks like Spatter evaluate the performance of new architectures for existing memory-bound application codes, without source-code optimization.

Organization

In this repository, there are 3 top-level directories: `benchmarks`, `build`, and `common_files`. All benchmark problems are inside the `benchmarks/<application>` directory. Job scripts and environment setup scripts are placed in the `common_files` directory to support reusability. Spatter source and the Spatter installation is in `build`.

Code Access and Compilation

The process of building the benchmark has three basic steps: obtaining the source code, configuring the build system, and compiling the source code. The build instructions in this section follows the `build_spatter_cpu.sh` script, which is suitable for a generic Linux workstation without a GPU accelerator. Detailed instructions for compiling the code on OLCF's Summit supercomputer using GPUs can be found in the README of the `build` directory. To use the provided shell scripts in `build`, modify the `common_files/environment /setup_env_{cpu,gpu}.sh` files to the correct modules and versions for your system. The provided environment is fit for Summit as of August 2023.

Obtaining the Spatter source code

The following commands were used to obtained the version of Spatter used provided here.

```
git clone https://github.com/hpcgarage/spatter.git
cd spatter/
git checkout 8f6384a
```

Configuring the Spatter build system

Spatter uses CMake to configure the build system and generate GNU makefiles. The below example compiles Spatter with the Serial CPU backend with a GCC compiler, and is identical to what is provided in `build/build_spatter_cpu.sh`.

```
cmake -DCMAKE_BUILD_TYPE=Release \
    -DBACKEND=Serial \
    -DCOMPILER=gnu \
    -DCMAKE_C_COMPILER=gcc \
    -DCMAKE_CXX_COMPILER=g++ ..
```

Spatter configuration used in the Summit benchmark

Software Prerequisites

- A supported C++11 compiler - GCC or Clang
- CMake 3.18+
- GNU make
- Optional: CUDA version 11.1 or greater

- Optional: OpenMP 3.0+ support

How to Build

Please use the script provided in `build_spatter_gpu.sh` to build the GPU-enabled binary of Spatter on Summit. This script sources the environment set up in `common_scripts/environment/setup_env_gpu.sh`.

Porting instructions

Spatter currently provides instructions for creating a new backend: https://github.com/hpcgarage/spatter/wiki/Adding-New-Backends-to-Spatter (https://github.com/hpcgarage/spatter/wiki/Adding-New-Backends-to-Spatter). For consistency, we provide the instructions from this link below with modification specific to OLCF-6 run rules.

1. Create a new backend directory using an existing directory as a template. For example, to begin adding a HIP backend, `cp -rf cuda hip`.
2. Rename the kernel files and edit them to add your new accelerator kernels for Scatter and Gather: `mv cuda-kernels.c hip-kernels.c'.
3. Add supporting code for allocating device memory and setup/teardown mechanisms to `<backend>/<backend>-backend.{h|c}`. As an example, CUDA setup/teardown code resides in `cuda/cuda-backend.h/c`.
4. Update the main header portion of main.c to add your backend's header files and supporting headers like cuda.h or omp.h.
5. Add your new backend's initialization code under "Initialization" section in main.c
6. Update the "Benchmark execution" section of code under main.c with ifdefs for your specific backend.
7. Add a new function for your backend into src/backend-support-tests.c and include/backend-support-tests.h to allow the main benchmark to easily check if your backend is enabled and compiled.
8. Update src/parse-args.c and include/parse-args.h to allow checks for your new backend.
9. Add a new configure script to build your particular backend in `configure/configure_<backend>`.
10. Update CMakeLists.txt to add support for finding your new backend's libraries and includes. Specifically, add the flag -DUSE_ and include your new backend's files in the compilation process.
11. Test your new branch using the configure script.
12. Create a new run script to run Spatter with your backend. (Copy a set of tests).

All new backends are required to perform the specified memory access patterns without modification or reordering.

Running the benchmark

Input files and batch scripts are provided for both represented applications in the `benchmarks` directory. Each subdirectory within the benchmarks directory contains a JSON file of 1 or more memory patterns derived from a specific application.

Each directory provides a symbolic link to the shared job script used to launch Spatter on Summit. This job script launches Spatter using command-line arguments provided by the `spatter_config_{cpu|gpu}.txt` file. CPU and GPU-enabled runs may be parameterized differently, so different command-line arguments and JSON files may be required.

Results

The figure of merit (FOM) is the achieved bandwidth, in megabytes per second (MB/s) (1 MB = 1,000,000 bytes).

Reference performance on Summit

Performance on Summit is to be provided.

Reporting

Benchmark results should include projections of the FOM for the provided memory access patterns. The complete hardware configuration needed to achieve the estimated timings must also be provided. Each memory pattern must be run by the same Spatter binary in the same environment. For the electronic submission, include all the source, build, and run files used to build and run on the target platform. Include all standard output files.